Japanese Patent Office

Patent Laying-Open Gazette

Patent Laying-Open No.	11-312152
Date of Laying-Open:	November 9, 1999
International Class(es):	G 06 F 15/16 H 04 Q 7/38

(28 pages in all)

Title of the Invention:	Mobile Electronic Device and Controlling Method Thereof
Patent Appln. No.	10-365666
Filing Date:	December 22, 1997
Priority Claimed:	Country: U.S.A. Filing Date: December 22, 1997 Serial No. 995606
Inventor(s):	Michael Mcmahon Marion C. Lineberry
Applicant(s):	Texas Instruments Inc.

(transliterated, therefore the spelling might be incorrect)

(2)

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平11－312152

(43)公開日　平成11年(1999)11月９日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　15/16	６２０	Ｇ０６Ｆ　15/16	６２０Ｇ
Ｈ０４Ｑ　7/38		Ｈ０４Ｂ　7/26	１０９Ｍ

審査請求　未請求　請求項の数２　ＯＬ　（全 28 頁）

(21)出願番号　特願平10－365666

(22)出願日　平成10年(1998)12月22日

(31)優先権主張番号　９９５６０６

(32)優先日　1997年12月22日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　590000879
テキサス　インスツルメンツ　インコーポレイテツド
アメリカ合衆国テキサス州ダラス，ノース　セントラルエクスプレスウエイ　13500

(72)発明者　マイケル　マクマホン
アメリカ合衆国　テキサス州プラノ，メリアン　ドライブ　3817

(72)発明者　マリオン　シー．ラインバリイ
アメリカ合衆国　テキサス州ダラス，エヌ．セルバ　ドライブ　1250

(74)代理人　弁理士　浅村　皓　（外３名）

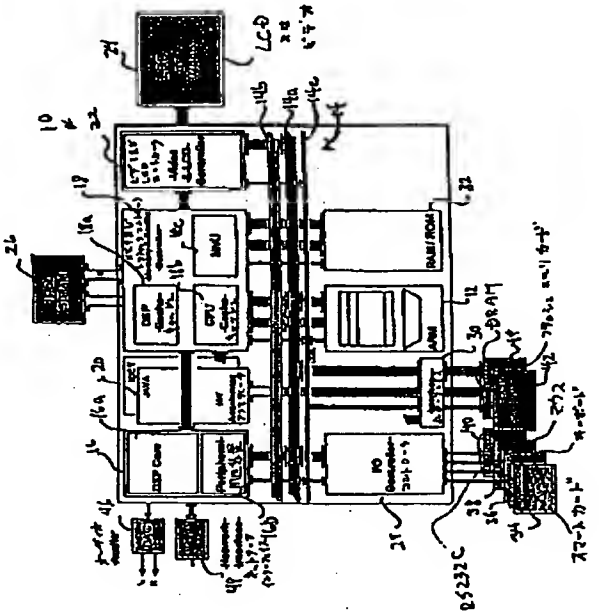
最終頁に続く

(54)【発明の名称】　移動電子装置および制御方法

(57)【要約】

【課題】　多数のプロセッサおよびコプロセッサの使用をグレードアップし最適化するデータ処理アーキテクチュアを得る。

【解決手段】　複数個のプロセッサ１２，１６を含むワイヤレスデータプラットホーム１０。タスクが実施される時に情報を通信できるようにプロセッサ間に通信チャネルが開設される。１個のプロセッサで実施されるダイナミッククロスコンパイラー８０がコードを別のプロセッサのためのネーティブ処理コードへコンパイルする。ダイナミッククロスリンカー８２が他のプロセッサのためにコンパイルされたコードをリンクする。ネーティブコードはそれを包むＪＡＶＡ　Ｂｅａｎ９０（もしくは他の言語タイプ）を使用してプラットホームへダウンロードすることもできる。ＪＡＶＡ　Ｂｅａｎは暗号化しセキュリティのためにデジタルサインすることができる。



【特許請求の範囲】
【請求項１】　移動電子装置であって、
ネィティブコードを実行するコプロセッサと、
ホストプロセッサシステムおよびプロセッサ独立コードに対応するネィティブコードを実行するように作動するホストプロセッサシステムであって、デジタル信号コプロセッサにより実施されるタスクをダイナミックに変化させるように作動するホストプロセッサシステムと、
前記ホストプロセッサシステムと前記コプロセッサ間の通信回路と、
を含む移動電子装置。
【請求項２】　移動電子装置の制御方法であって、
コプロセッサにおいてネィティブコードを実行するステップと、
ホストプロセサシステムにおいてネィティブコードおよびプロセッサ独立コードを実行するステップと、
デジタル信号コプロセッサにより実施されるタスクを前記ホストプロセサシステムによりダイナミックに変えるステップと、
前記ホストプロセッサシステムと前記コプロセッサ間で通信を行うステップと、
を含む方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は一般的に移動電子装置に関し、特に移動電子装置用ハードウェアおよびソフトウェアプラットホームに関する。
【０００２】
【従来の技術】装置のパワー、したがって機能、が増強されるにつれてハンドヘルド携帯装置の人気が高まってきている。近年ＰＤＡ（Ｐｅｒｓｏｎａｌ　ＤｉｇｉｔａｌＡｓｓｉｓｔａｎｔ）が広く使用されており、セルラー電話機とＰＤＡのいくつかの能力を組み合わせたスマートホーン（Ｓｍａｒｔ　ｐｈｏｎｅｓ）が近未来の通信に著しいインパクトを与えるものと考えられる。
【０００３】現在ある装置には、音声認識等の、ある個別の機能を提供する１個以上のＤＳＰ（デジタル信号処理装置）もしくは他のコプロセッサ、および他のデータ処理機能のための汎用プロセッサが内蔵されている。ＤＳＰ用コードおよび汎用プロセッサ用コードは一般的にＲＯＭその他の非揮発性メモリに格納され、それらは容易に修正することができない。したがって、改善および新しい機能を利用できるようになった時に、しばしば装置の能力をグレードアップできないことがある。特に、装置内に存在することがあるＤＳＰや他のコプロセッサを最大限に使用することができない。
【０００４】
【発明が解決しようとする課題】したがって、多数のプロセッサおよびコプロセッサの使用をグレードアップし最適化することができるデータ処理アーキテクチュアが必要とされている。
【０００５】
【課題を解決するための手段】本発明において、移動電子装置はネィティブコードを実行するコプロセッサと、ホストプロセッサシステムに対応するネィティブコードおよびプロセッサ独立コードを実行するように作動するホストプロセッサシステムとを含んでいる。ホストプロセッサシステムはデジタル信号コプロセッサにより実施されるタスクをダイナミックに変えるように作動する。通信回路によりホストプロセッサシステムとコプロセッサ間の通信が提供される。
【０００６】本発明により従来技術を凌ぐ著しい利点が提供される。それは、ホストプロセッサシステムが、デジタル信号処理装置であっても良いコプロセッサにより実施されるタスクをダイナミックに割り当ててコプロセッサを完全に使用できるためである。ホストプロセッサシステムは、各プロセッサの現在の能力等の、多様な要因に応じて複数のコプロセッサの１つへ直接ルーチンすることができる。
【０００７】
【発明の実施の形態】図１に例えばスマートホーンやＰＤＡを実施するのに使用できる一般的なワイヤレスデータプラットホームアーキテクチュアの好ましい実施例を示す。ワイヤレスデータプラットホーム１０は、データバス１４ａ、アドレスバス１４ｂおよびコントロールバス１４ｃを含むバス構造１４に接続された汎用（ホスト）プロセッサ１２を含んでいる。コアプロセッサ１６ａおよび周辺インターフェイス１６ｂを含む１個以上のＤＳＰ（もしくは他のコプロセッサ）１６がバス１４およびＤＳＰキャッシュメモリ１８ａ、ＣＰＵキャッシュ１８ｂ、およびＭＭＵ（メモリマネジメントユニット）１８ｃを含むメモリおよびトラフィックコントローラ１８に接続されている。ハードウェアアクセラレータ回路２０（ＪＡＶＡ等のポータブル言語を加速する）およびビデオおよびＬＣＤコントローラ２２もメモリおよびトラフィックコントローラ１８に接続されている。ビデオおよびＬＣＤコントローラの出力はＬＣＤもしくはビデオディスプレイ２４に接続されている。
【０００８】メモリおよびトラフィックコントローラ１８はバス１４および、ＳＤＲＡＭ（同期ダイナミックランダムアクセスメモリ）として示す、主記憶装置２６に接続されている。バス１４はＩ／Ｏコントローラ２８、インターフェイス３０、およびＲＡＭ／ＲＯＭ３２にも接続されている。複数の装置をスマートカード３４、キーボード３６、マウス３８等のワイヤレスデータプラットホーム１０もしくはＵＳＢ（ユニバーサルシリアルバス）ポートもしくはＲＳ２３２シリアルポート等の１つ以上のシリアルポート４０に接続することができる。インターフェイス３０はフラッシュメモリカード４２および／もしくはＤＲＡＭカード４４に接続することができ

る。周辺インターフェイス１６ｂはＤＳＰ１６をＤＡＣ（デジタル／アナログコンバータ）４６、ネットワークインターフェイス４８もしくは他の装置に接続することができる。

【０００９】図１のワイヤレスデータプラットホーム１０は汎用プロセッサ１２およびＤＳＰ１６を利用している。ＤＳＰ１６が特定の固定機能専用とされている現在の装置とは異なり、図１のＤＳＰ１６は任意数の機能に使用することができる。それにより、ユーザはＤＳＰ１６の利益を完全に引き出すことができる。

【００１０】ＤＳＰ１６を使用できる１つの主要なエリアはマンーマシンインターフェイス（ＭＭＩ）に関連している。重要なのは、音声認識、イメージおよびビデオ圧伸、データ暗号化、テキストー音声変換、等の機能をＤＳＰ１６を使用してより効率的に実施できることである。本アーキテクチュアにより新しい機能や改善をワイヤレスデータプラットホーム１０へ容易に付加することができる。

【００１１】ワイヤレスデータプラットホーム１０は一般的なブロック図であり、さまざまに修正することができる。例えば、図１には独立したＤＳＰおよびプロセッサキャッシュ１８ａ，１８ｂが図示されている。当業者にはお判りのように、ユニット型キャッシュも使用することができる。さらに、ハードウェアアクセラレーション（ａｃｃｅｌｅｒａｔｉｏｎ）回路２０はオプショナルアイテムである。このような装置によりＪＡＶＡ等の言語の実行が加速されるが、この回路は装置の動作に必要なものではない。さらに、１個のＤＳＰしか図示されていないが、多数のＤＳＰ（もしくは他のコプロセッサ）をバスに接続することができる。

【００１２】図２にワイヤレスデータプラットホーム１０の機能的ソフトウェアアーキテクチュアを示す。このブロック図はＪＡＶＡを使用するものと想定しており、ＪＡＶＡ以外の言語も使用することができる。機能的に、ソフトウェアは２つのグループへ分割され、それはホストプロセッサソフトウェアおよびＤＳＰソフトウェアである。ホストソフトウェアは１つ以上のアプレット４０を含んでいる。ＤＳＰ　ＡＰＩクラス４２はＪＡＶＡアプリケーション用ＪＡＶＡ　ＡＰＩパッケージ、すなわちＤＳＰ　ＡＰＩ５０およびホストＤＳＰインターフェイスレイヤ５２の機能へアクセスするアプレットである。ＪＡＶＡバーチャルマシン（ＶＭ）４４がアプレットを解釈する。ＪＡＶＡネイティブインターフェイス４６は、ＪＡＶＡ　ＶＭがホストプロセッサもしくは特定プラットホームコードを実行する方法である。ネイティブタスク４８はＪＡＶＡネイティブインターフェイスを使用せずにホストプロセッサ１２により実行することができる非ＪＡＶＡプログラムである。後述するＤＳＰ　ＡＰＩ５０は、ＤＳＰ１６の能力を使用するためにコールするホスト１２で使用されるＡＰＩ（アプリケーションプログラムインターフェイス）である。ホストーＤＳＰインターフェイスレイヤ５２は他のタスク、あるいはホストーＤＳＰ通信プロトコルを介したチャネルを使用して他のハードウェアと互いに通信するためのＡＰＩをホスト１２およびＤＳＰ１６に提供する。ＤＳＰデバイスドライバ５４はＤＳＰ１６と通信するためのホストＲＴＯＳ５６（リアルタイムオペレーティングシステム）用ホストベースデバイスドライバである。ホストＲＴＯＳ５６はアクセラレーテッドテクノロジー社のＮＵＣＬＥＵＳ　ＰＬＵＳ等のオペレーティングシステムである。マイクロソフト社のＷＩＮＤＯＷＳＣＥ等の非リアルタイムオペレーティングシステムを使用することもできる。ＤＳＰライブラリ５８にはＤＳＰ１６で実行するために格納されたプログラムが含まれている。

【００１３】ＤＳＰ側で、ＤＳＰ１６により実行する１つ以上のタスク６０をメモリ内に格納することができる。後述するように、ＤＳＰの機能がスタティックではなくダイナミックとなるように、タスクを所望によりメモリから出し入れすることができる。ＤＳＰ側のホストーＤＳＰインターフェイスレイヤ６２はホスト側のホストーＤＳＰインターフェイスレイヤ５２と同じ機能を実施する、すなわちホスト１２およびＤＳＰ１６は通信することができる。ＤＳＰ　ＲＴＯＳ６４はＤＳＰプロセッサ用オペレーティングシステムである。ホストデバイスドライバ６６はホスト１２と通信するＤＳＰ　ＲＴＯＳ６４用ＤＳＰベースデバイスドライバである。ホストーＤＳＰインターフェイス７０はＤＳＰ１６をホスト１２に接続する。

【００１４】動作において、図２に示すソフトウェアアーキテクチュアはＤＳＰ１６を従来技術のように固定機能装置ではなく、可変機能装置として使用する。したがって、ＤＳＰ機能は図２のアーキテクチュアを内蔵する移動装置へダウンロードして、ＤＳＰ１６がホスト１２に対してさまざまな信号処理機能を実施できるようにすることができる。

【００１５】ＤＳＰーＡＰＩ

ＤＳＰーＡＰＩはホスト１２からＤＳＰ１６へのデバイス独立インターフェイスを提供する。この機能により、ホスト１２にはＤＳＰ１６にタスクをロードしてスケジュールを行いかつこれらのタスクをコントロールして通信する能力が与えられる。ＡＰＩ機能には、ＤＳＰの利用可能な資源を決定し、ホスト１２およびＤＳＰタスクを生成かつコントロールし、ホスト１２およびＤＳＰタスク間のデータチャネルを生成かつコントロールし、タスクと通信するための呼出しが含まれる。これらの機能については後述する。各機能はブール結果を戻し、それは成功したオペレーションに対するＳＵＣＣＥＳＳ、もしくはＦＡＩＬＵＲＥである。結果がＦＡＩＬＵＲＥであれば、ｅｒｒｃｏｄｅをチェックしてどのエラーが発生したかを確認しなければならない。

【００１６】

DSP_Get_MIPS

*BOOL DSP_Get_MIPS(T_DeviceID DevID, U32 *mips, U16 *errcode);*

この関数はＤＳＰ上で利用できる現在のＭＩＰＳを戻す。それはＤＳＰ１６のＭＩＰＳ能力マイナスベースＭＩＰＳ値（付加ダイナミックタスクの無いＭＩＰＳ値、すなわちカーネルプラスＡＰＩコードプラスドライバ）、マイナスロードされた全てのダイナミックタスクに対するＭＩＰＳレーティングの和である。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数１】

【数２】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

【００１７】

【数３】

DSP_Get_Memory_Available

*BOOL DSP_Get_Memory_Available(T_DeviceID DevID, T_Size *progmem, T_Size *datamem, U16 *errcode);*

この関数はプログラムメモリおよびデータメモリの両方に対して利用できるメモリ量についてＤｅｖＩＤにより指定されるＤＳＰ１６に質問する。その結果得られる値はｐｒｏｇｍｅｍおよびｄａｔａｍｅｍパラメータ内に戻される。サイズはＴ＿ＤＳＰ＿Ｗｏｒｄｓ内に指定される。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数４】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

【００１８】

【数５】

DSP_Alloc_Mem

*BOOL DSP_Alloc_Mem(T_DeviceID DevID, U16 mempage, T_Size size, T_DSP_Word **memptr, U16 *errcode);*

この関数によりＤＳＰ１６上に１ブロックのメモリが割り当てられる。ＤｅｖＩＤはどのデバイス上にメモリを割り当てるかを指定する。ｍｅｍｐａｇｅはプログラムスペースに対しては０であり、データスペースに対しては１である。ｓｉｚｅパラメータはＴ＿ＤＳＰ＿Ｗｏｒｄｓ内のメモリブロックサイズを指定する。戻されるｍｅｍｐｔｒはＤＳＰ１６上のメモリブロックへのポインター、もしくは失敗時のＮＵＬＬである。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数６】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_NOT_ENOUGH_MEMORY

【００１９】

【数７】

DSP_Free_Mem

*BOOL DSP_Free_Mem(T_DeviceID DevID, U16 mempage, T_DSP_Word *memptr, U16 *errcode);*

この関数はＤＳＰ
Ａｌｌｏｃ＿Ｍｅｍ関数を割り当てられたＤＳＰ上の１ブロックのメモリを解放する。ＤｅｖＩＤはどのデバイス上にメモリが常駐するかを指定する。ｍｅｍｐａｇｅはプログラムスペースに対しては０であり、データスペースに対しては１である。ｍｅｍｐｔｒパラメータはメモリブロックへのポインターである。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数８】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_MEMBLOCK_NOT_FOUND

【００２０】

【数９】

DSP_Get_Code_Inf

*BOOL DSP_Get_Code_Info(char *Name, T_CodeHdr *codehdr, U16 *errcode);*

この関数はＤＳＰライブラリテーブルへアクセスし、Ｎａｍｅパラメータにより指定されるＤＳＰ関数コードのコードヘッダーを戻す。戻されると、ｃｏｄｅｈｄｒパラメータにより指示される位置はコードヘッダー情報を含む。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数１０】

DSP_SUCCESS
DSP_NAMED_FUNC_NOT_FOUND

【００２１】

【数１１】

DSP_Link_Code

*BOOL DSP_Link_Code(T_DeviceID DevID, T_CodeHdr *codehdr, T_TaskCreate *tcs, U16 *errcode);*

この関数はＤｅｖＩＤにより指定されるＤＳＰ上の指示されたアドレスで走るようにＤＳＰ関数コードをリンクする。ｃｏｄｅｈｄｒパラメータは関数のコードヘッダーを指示する。ダイナミッククロスリンカーはコードヘッダー内およびコード（ＣＯＦＦファイル）内の情報に基づいてコードをリンクする。ダイナミッククロスリンカーは必要に応じてメモリを割り当て、ＤＳＰ１６にコードをリンクしロードする。ｔｃｓパラメータはＤＳＰ＿Ｃｒｅａｔｅ＿Ｔａｓｋ関数において必要なタスク生成構造へのポインターである。ＤＳＰ＿Ｌｉｎｋ＿Ｃｏｄｅはタスクの生成に備えて構造のコードエントリーポイント、優先順位、およびクオンタム（ｑｕａｎｔｕｍ）フィールドを充填する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数１２】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_NOT_ENOUGH_PROG_MEMORY
DSP_NOT_ENOUGH_DATA_MEMORY
DSP_COULD_NOT_LOAD_CODE

【００２２】

【数１３】

DSP_Put_BLOB

*BOOL DSP_Put_BLOB(T_DeviceID DevID, T_HostPtr srcaddr, T_DSP_Ptr destaddr, U16 mempage, T_Size size, U16 *errcode);*

この関数は指定されたＢＬＯＢ（Ｂｉｎａｒｙ Ｌａｒｇｅ Ｏｂｊｅｃｔ）をＤＳＰ１６へコピーする。ＤｅｖＩＤはどのＤＳＰ１６へオブジェクトをコピーするかを指定する。ｓｒｃａｄｄｒパラメータはホストメモリ内のオブジェクトへのポインターである。ｄｅｓｔａｄｄｒはＤＳＰ１６上のオブジェクトをコピーする位置へのポインターである。ｍｅｍｐａｇｅはプログラムスペースに対しては０であり、データスペースに対しては１である。サイズパラメータはＴＤＳＰ＿Ｗｏｒｄｓ内のオブジェクトのサイズを指定する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数１４】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

【００２３】

【数１５】

DSP_Create_Task

*BOOL DSP_Create_Task(T_DeviceID DevID, T_TaskCreate *tcs, T_TaskID *TaskID, U16 *errcode);*

ＤＳＰ Ｃｒｅａｔｅ＿Ｔａｓｋは、タスクパラメータおよびＤＳＰのプログラムスペース内のコード位置が与えられたらタスクを生成するようＤＳＰ１６に要求する。タスク生成構造を表１に示す。

【００２４】

【表１】

表１　タスク生成構造

データタイプ	フィールド名	説明
T_DSP_Name	Name	User defined name for the task. （タスクのユーザ定義名）
U32	MIPS	MIPS used by the task. （タスクが使用するＭＩＰＳ）
T_ChanID	ChanIn	The channel ID used for task input. （タスク入力に使用するチャネルＩＤ）
T_ChanID	ChanOut	The channel ID used for task output. （タスク出力に使用するチャネルＩＤ）
T_StrmID	StrmIn	The stream ID used for task input. （タスク入力に使用するストリームＩＤ）
T_StrmID	StrmOut	The stream ID used for task output. （タスク出力に使用するストリームＩＤ）
U16	Priority	The task's priority. （タスクの優先順位）
U32	Quantum	The task's timeslice in system ticks. （システムチック内のタスクのタイムスライス）
T_Size	StackReq	The amount of stack required. （所有タスク量）
T_DSP_Ptr	MsgHandler	Pointer to code to handle messages to the task. （タスク へのメッセージ を処理するコード へのポインター ）
T_HOST_Ptr	CallBack	Pointer to Host code to handle messages from the task. （タスク からのメッセージ を処理するホストコードへのポインター ）
T_DSP_Ptr	Create	Pointer to code to execute when task is created. （タスク生成時に実行するコードへのポインター）
T_DSP_Ptr	Start	Pointer to code to execute when task is started. （タスク開始時に実行するコードへのポインター）
T_DSP_Ptr	Suspend	Pointer to code to execute when task is suspended. （タスク保留時に実行するコードへのポインター）
T_DSP_Ptr	Resume	Pointer to code to execute when task is resumed. （タスク再開時に実行するコードへのポインター）
T_DSP_Ptr	Stop	Pointer to code to execute when task is stopped. （タスク停止時に実行するコードへのポインター）

【００２５】タスクが生成されると、Ｃｒｅａｔｅエントリーポイントが呼び出され、任意の必要な予備初期化を行う機会をタスクに与える。Ｃｒｅａｔｅ，Ｓｕｓｐｅｎｄ，Ｒｅｓｕｍｅ，およびＳｔｏｐエントリーポイントはＮＵＬＬとすることができる。その結果得られるＴａｓｋＩＤはデバイスＩＤ（ＤｅｖＩＤ）とＤＳＰのタスクＩＤの両方を含んでいる。ＴａｓｋＩＤがＮＵＬＬであれば、生成は失敗である。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数１６】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_PRIORITY
DSP_CHANNEL_NOT_FOUND
DSP_ALLOCATION_ERROR

【００２６】

【数１７】

DSP_Start_Task

*BOOL DSP_Start_Task(T_TaskID TaskID, U16 *errcode);*

この関数はＴａｓｋＩＤにより指定されるＤＳＰタスクを開始する。タスクのＳｔａｒｔエントリーポイントで実行が開始する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【００２７】

【数１９】

【数１８】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Suspend_Task

*BOOL DSP_Suspend_Task(T_TaskID TaskID, U16 *errcode);*

この関数はＴａｓｋＩＤにより指定されたＤＳＰタスクを保留する。保留される前に、任意の必要なハウスキーピングを実施する機会をタスクに与えるためにタスクのＳｕｓｐｅｎｄエントリーポイントが呼び出される。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

【００２８】

【数２０】

【数２１】

DSP_Resume_Task

*BOOL DSP_Resume_Task(T_TaskID TaskID, U16 *errcode);*

この関数はＤＳＰ Ｓｕｓｐｅｎｄ＿Ｔａｓｋにより保留されたＤＳＰタスクを再開する。再開される前に、任意の必要なハウスキーピングを実施する機会をタスクに与えるためにタスクのＲｅｓｕｍｅエントリーポイントが呼び出される。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_TASK_NOT_SUSPENDED

【００２９】

【数２２】

【数２３】

DSP_Delete_Task

*BOOL DSP_Delete_Task(T_TaskID TaskID, U16 *errcode);*

この関数はＴａｓｋＩＤにより指定されたＤＳＰタスクを削除する。削除する前に、任意の必要なクリーンナップを実施する機会をタスクに与えるためにタスクのＳｔｏｐエントリーポイントが呼び出される。それはタスクにより割り当てられた任意のメモリを解放し、タスクが取得した任意の資源を戻すことを含まなければならない。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数２４】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

【００３０】

【数２５】

DSP_Change_Task_Priority

*BOOL DSP_Change_Task_Priority(T_TaskID TaskID, U16 newpriority, U16 *oldpriority, U16 *errcode);*

この関数はＴａｓｋＩＤにより指定されたＤＳＰタスクの優先順位を変える。優先順位はｎｅｗｐｒｉｏｒｉｔｙへ変えられる。ｎｅｗｐｒｉｏｒｉｔｙの考えられる値はＲＴＯＳ依存である。戻ると、ｏｌｄｐｒｉｏｒｉｔｙパラメータはタスクの前の優先順位に設定される。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_INVALID_PRIORITY

【００３１】

【数２６】

【数２７】

DSP_Get_Task_Status

*BOOL DSP_Get_Task_Status(T_TaskID TaskID, U16 *status, U16 *priority, T_ChanID *Input, T_ChanID Output, U16 *errcode);*

この関数はＴａｓｋＩＤにより指定されたＤＳＰタスクの状態を戻す。ｓｔａｔｕｓは下記の値の１つを含む。

【数２８】

DSP_TASK_RUNNING
DSP_TASK_SUSPENDED
DSP_TASK_WAITFOR_SEM
DSP_TASK_WAITFOR_QUEUE
DSP_TASK_WAITFOR_MSG

【００３２】ｐｒｉｏｒｉｔｙパラメータはタスクの優先順位を含み、ＩｎｐｕｔおよびＯｕｔｐｕｔパラメータはタスクの、それぞれ、入力および出力ＩＤを含む。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数２９】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

DSP_Get_ID_From_Name

*BOOL DSP_Get_ID_From_Name(T_DeviceID DevID, T_DSP_Name Name, T_DSP_ID *ID, U16 *errcode);*

【００３３】

【数３０】

この関数はＤＳＰ１６上の名前を付けたオブジェクトのＩＤを戻す。名前を付けたオブジェクトはチャネル、タスク、メモリブロック、もしくは任意他のサポートされた名前を付けたＤＳＰオブジェクトとすることができる。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数３１】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_NAME_NOT_FOUND

【００３４】

【数３２】

DSP_Dbg_Read_Mem

*BOOL DSP_Dbg_Read_Mem(DEVICE_ID DevID, U8 mempage, DSP_PTR addr, U32 count, DSP_WORD *buf, U16 *errcode);*

この関数はメモリの１ブロックを要求する。ｍｅｍｐａｇｅはプログラムメモリ（０）もしくはデータメモリ（１）を指定する。ａｄｄｒパラメータはメモリ開始アドレスを指定し、ｃｏｕｎｔはどれだけ多くのＴ＿ＳＤＰ＿Ｗｏｒｄｓを読み出すかを示す。ｂｕｆパラメータはメモリをコピーしなければならないバッファーを提供した呼び出し者へのポインターである。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数３３】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

【００３５】

【数３４】

DSP_Dbg_Write_Mem

*BOOL DSP_Dbg_Write_Mem(T_DeviceID DevID, U16 mempage, T_DSP_Ptr addr, T_Count count, T_DSP_Word *buf, U16 *errcode);*

この関数はメモリの１ブロックを書き込む。ｍｅｍｐａｇｅはプログラムメモリ（０）もしくはデータメモリ（１）を指定する。ａｄｄｒパラメータはメモリ開始アドレスを指定し、ｃｏｕｎｔはどれだけ多くのＴ＿ＳＤＰ＿Ｗｏｒｄｓを書き込むかを示す。ｂｕｆパラメータは書き込むメモリを含むバッファーへのポインターである。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数３５】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

【００３６】

【数３６】

DSP_Dbg_Read_Reg

*BOOL DSP_Dbg_Read_Reg(T_DeviceID DevID, U16 RegID, T_DSP_Word *regvalue, U16 *errcode);*

この関数はＤＳＰレジスタを読み出しｒｅｇｖａｌｕｅ内の値を戻す。ＲｅｇＩＤパラメータはどのレジスタを戻すべきかを指定する。ＲｅｇＩＤが－１であれば、全てのレジスタ値が戻される。バッファーを提供した発呼

者へのポインターであるｒｅｇｖａｌｕｅパラメータは、全ての値を保持するのに十分な記憶装置を指示しなければならない。レジスタＩＤは特定ＤＳＰ向けであり特定のインプリメンテーションに依存する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数３７】

DSP_Dbg_Write_Reg

*BOOL DSP_Dbg_Write_Reg(T_DeviceID DevID, U16 RegID, T_DSP_Word regvalue, U16 *errcode);*

この関数はＤＳＰレジスタに書き込む。ＲｅｇＩＤパラメータはどのレジスタを修正するかを指定する。ｒｅｇｖａｌｕｅは書き込む新しい値を含んでいる。レジスタＩＤは特定ＤＳＰ向けであり特定のインプリメンテーションに依存する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数３９】

DSP_Dbg_Set_Break

*BOOL DSP_Dbg_Set_Break(T_DeviceID DevID, DSP_Ptr addr, U16 *errcode);*

この関数は所与のコードアドレス（ａｄｄｒ）におけるブレイクポイント（ｂｒｅａｋ　ｐｏｉｎｔ）を設定する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数４１】

DSP_Dbg_Clr_Break

*BOOL DSP_Dbg_Clr_Break(T_DeviceID DevID, T_DSP_Ptr addr, U16 *errcode);*

この関数は所与のコードアドレス（ａｄｄｒ）においてＤＳＰ
Ｄｂｇ＿Ｓｅｔ＿Ｂｒｅａｋにより予め設定されたブレイクポイントをクリアする。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数４３】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BP_DID_NOT_EXIST

【００４０】ＤＳＰデバイスドライバ
ＤＳＰデバイスドライバ５４はホスト１２からＤＳＰ１６への通信を処理する。ドライバ関数はホストーＤＳＰ通信プロトコルに明記された通信要求を取り上げ、利用可能なハードウェアインターフェイスを介した情報の伝送を処理する。デバイスドライバはＲＴＯＳ依存かつ通信ハードウェア依存である。

【００４１】ＤＳＰライブラリ
ＤＳＰライブラリ５８はＤＳＰ１６へダウンロードして実行することができるコードのブロックを含んでいる。

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER

【００３７】

【数３８】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER

【００３８】

【数４０】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

【００３９】

【数４２】

ダイナミッククロスリンカーが全てのアドレス参照（ｒｅｆｅｒｅｎｃｅ）を決定できるように、コードの各ブロックは予め非リンク、すなわちライブラリとして再配置可能にリンクされている。各コードブロックはＤＳＰＭＩＰＳ（百万命令／秒）、優先順位、タイムスライスクオンタム、およびメモリに対するブロックの要求に関する情報も含んでいる。コードブロックヘッダーのフォーマットを表２に示す。プログラムメモリおよびデータメモリサイズは、ＤＳＰがタスクのメモリ要求をサポートできるかどうかの迅速なチェックをホスト１２に与えるための近似値である。十分なスペースがあるようであれば、ダイナミッククロスリンカーはコードのリンクおよびロードを試みることができる。ページアライメント（ａｌｉｇｎｍｅｎｔ）および連続性（ｃｏｎｔｉｇｕｉｔｙ）の要求により、ダイナミッククロスリンカーはそれでも目的を果たせないことがある。好ましい実施例では、コードはバージョン２ＣＯＦＦファイルフォーマットである。

【００４２】

【表２】

表２　コードブロックヘッダー

データタイプ	フィールド名	説明
U16	Processor	The target processor type. （ターゲットプロセッサタイプ）
T_DSP_Name	Name	Task's name. （タスク名）
U32	MIPS	Worst case MIPS required by the task. （タスクが要求する最悪時のＭＩＰＳ）
T_Size	ProgSize	Total program memory size needed. （所用総プログラムメモリサイズ）
T_Size	DataSize	Total data memory size needed. （所用総データメモリサイズ）
T_Size	InFrameSize	Size of a frame in the task's input channel. （タスク入力チャネル内のフレームサイズ）
T_Size	OutFrameSize	Size of a frame in the task's output channel. （タスク出力チャネル内のフレームサイズ）
T_Size	InStrmSize	Size of the task's input stream FIFO. （タスクの入力ストリームＦＩＦＯのサイズ）
T_Size	OutStrmSize	Size of the task's output stream FIFO. （タスクの出力ストリームＦＩＦＯのサイズ）
U16	Priority	Task's priority. （タスクの優先順位）
U32	Quantum	Task's time slice quantum (number of system ticks). （タスクのタイムスライスクォンタム）（システムチック数）
T_Size	StackReq	Stack required. （所用スタック）
T_Size	CoffSize	Total size of the COFF file. （ＣＯＦＦファイルの総サイズ）
T_DSP_Ptr	MsgHandler	Offset to a message handler entry point for the task. （タスクのメッセージハンドラーエントリポイントへオフセット）
T_DSP_Ptr	Create	Offset to a create entry point that is called when the task is created. （タスク生成時に呼び出される生成エントリポイントへオフセット）
T_DSP_Ptr	Start	Offset to the start of the task's code. （タスクコードの開始へオフセット）
T_DSP_Ptr	Suspend	Offset to a suspend entry point that is called prior to the task being suspended. （保留されるタスク よりも前に呼び出される保留エントリポイントへオフセット）
T_DSP_Ptr	Resume	Offset to a resume entry point that is called prior to the task being resumed. （再開されるタスク よりも前に呼び出される再開エントリポイントへオフセット）
T_DSP_Ptr	Stop	Offset to a stop entry point that is called prior to the task being deleted. （削除されるタスク よりも前に呼び出される停止エントリポイントへオフセット）
T_Host_Ptr	CoffPtr	Pointer to the location of the COFF data in the DSP Library. （ＤＳＰライブラリ内のＣＯＦＦデータ位置へのポインター）

【００４３】ポータブルコードのリンクされたターゲットコードへの変換

ＪＡＶＡコード等のポータブル（プロセッサ独立）コードをリンクされたターゲットコードへ変換する手順を図３に示す。この手順は２つの機能、ダイナミッククロスコンパイラー８０およびダイナミッククロスリンカー８２、を使用する。各機能はホストプロセッサ１２上で実現される。好ましい実施例では、ダイナミッククロスリンカーはＤＳＰ－ＡＰＩの一部である。クロスコンパイラーはＤＳＰ＿ＡＰＩの一部とすることもできる。

【００４４】ダイナミッククロスコンパイラー８０はポータブルコードをリンクされない、実行可能なターゲットプロセッサコードへ変換する。ダイナミッククロスリンカー８２は非リンク、実行可能なターゲットプロセッサコードをリンクされた、実行可能なターゲットプロセッサコードへ変換する。そうするために、ＤＳＰ１６上へローディングする前に、１ブロックのコード内のアドレスを決定しなければならない。ダイナミッククロスリンカー８２は関数のコードセグメントおよびデータセグメントをリンクし、ＤＳＰ１６上のメモリを割り当て、コードおよび一定のデータをＤＳＰ１６へロードする。コードを実行するターゲットプロセッサ（すなわち、ＤＳＰ１６）とは異なるプロセッサ（すなわち、ホストプロセッサ１２）で関数（コンパイリングおよびリンキング）が生じるため、この関数は“クロス”コンパイリングおよび“クロス”リンキングと言われる。

【００４５】ダイナミッククロスコンパイラー８０はユーザもしくはユーザエージェント（ブラウザ等）がオンデマンドでロードした予めリンクされていないコードを受け取る。コードは（１）コードの“タグ”部を識別するか、もしくは（２）ＤＳＰ１６で実行する適性について非タグコードセグメントを解析するために処理され

る。ソースコードのタグ部はその中に埋め込まれた“＜ｓｔａｒｔ　ＤＳＰ　ｃｏｄｅ＞”および“＜ｅｎｄ　ＤＳＰ　ｃｏｄｅ＞”等の所定のマーカによりＤＳＰへターゲット可能なソースを描くことができる。タグ部が直接もしくは解析により識別される場合には、ＤＳＰ１６の現在の処理状態に基づいてクロスコンパイルするかどうかが判断される。コンパイルすると判断されれば、コードのその部分は既知のコンパイリング方法を使用して非リンク、実行可能ターゲットプロセッサコードを出力するソフトウェアをコンパイリングすることにより処理される。コンパイルしないという判断は、例えば、ＤＳＰ１６により他のタスクが実行されているため、ＤＳＰが利用できる容量（一般的に、利用可能ＭＩＰＳ－百万命令／秒といわれる）が不十分であるか、利用可能なメモリが不十分である場合になされる。コンパイルされたコードはダイナミッククロスリンカー８２へ通してＤＳＰ１６において即座に使用するか、あるいはＤＳＰライブラリ５８内に保存することができる。

【００４６】ダイナミッククロスリンカー８２は予め非リンクコードを受け取り、それは（１）ホストプロセッサ１２と関連してスタティックに格納されるか、（２）ネットワークコネクション（インターネット等のグローバルネットワークを含む）を介してホストプロセッサ１２へダイナミックにダウンロードされるか、あるいは（３）ダイナミッククロスコンパイラー８０によりダイナミックに発生される。ダイナミッククロスリンカー８２はランタイムに決定されるＤＳＰ１６のメモリ開始アドレスの入力コードをリンクする。メモリ開始アドレスはホストプロセッサ１２もしくはＤＳＰ１６により格納され管理されるメモリマップもしくはメモリテーブルから決定することができる。ダイナミッククロスリンカー８２はコード内の基準メモリ位置をＤＳＰ内の実際のメモリ位置へ変換する。これらのメモリ位置は、例えば、コード内の分岐アドレスもしくはコード内のデータ位置参照（ｒｅｆｅｒｅｎｃｅ）を含むことができる。

【００４７】実施例では、ポータブルコードはリンクされているかどうかを含めたコードに関する全情報を含むＣＯＦＦ（共通オブジェクトファイルフォーマット）内にある。リンクされていなければ、コードをリンクするのに変えなければならないアドレスをシンボルテーブルが定義する。

【００４８】前記した変換プロセスには従来技術に較べていくつかの著しい利点がある。第１に、ダイナミッククロスコンパイラー８０はダウンロードされたポータブルコードをどこで実行するかについてランタイム判断を行うことができる。例えば、多数のターゲットプロセッサ（２台のＤＳＰ１６等）を有するシステムでは、ダイナミッククロスコンパイラー８０は利用可能な資源もしくは能力に基づいてポータブルコードを任意の１台のターゲットプロセッサへコンパイルすることができる。ダイナミッククロスリンカー８２はリロケータブルコードをサポートしないターゲットプロセッサでランするようコードをリンクする。コードはランタイムにリンクされるため、ＤＳＰ１６（もしくは、他のターゲットプロセッサ）内のメモリ位置は保存する必要がなく、デバイス内の全ての計算資源を最適効率で使用することができる。コンパイリングはプラットホーム１０のアーキテクチャの知識により達成されるため、一方もしくは両方のプロセッサのインテクジェントキャッシュアーキテクチュア等の特定プロセッサおよびプラットホーム向けの特徴をコンパイリングに利用することができる。

【００４９】したがって、ＤＳＰ１６はその処理能力を完全に使用するようにダイナミックに変えられるさまざまな機能を有することができる。例えば、ユーザは音声認識を含むユーザインターフェイスをロードしたいことがある。その時、ホストプロセッサはソフトウェアをダウンロードしてＤＳＰ１６で実行する音声認識ソフトウェアをダイナミックにクロスコンパイルおよびクロスリンクする。あるいは、ＤＳＰ１６の現在状態に基づいてＤＳＰライブラリ５８内の予めコンパイルされたソフトウェアをダイナミックにクロスリンクして実行することができる。

【００５０】ホストデバイスドライバ

ホストデバイスドライバはＤＳＰ１６からホスト１２への通信を処理する。ドライバ機能はホスト－ＤＳＰ通信プロトコルに明記された通信要求を取り入れて利用可能なハードウァエインターフェイスを介した情報の伝送を処理する。デバイスドライバはＲＴＯＳ依存かつ通信ハードウェア依存である。

【００５１】ホスト－ＤＳＰ通信プロトコル（ホスト－ＤＳＰインターフェイスレイヤ）

ホスト－ＤＳＰ通信プロトコルはホスト１２とＤＳＰ１６間のコマンドおよびデータの通信を支配する。通信はいくつかのパス、メッセージ、データチャネル、およびストリーム、からなる。メッセージは初期化パラメータおよびコマンドをタスクへ送るのに使用される。データチャネルはタスク間およびＤＳＰ１６とホスト１２間で大量のデータをデータフレームの形で運ぶ。ストリームはタスク間およびＤＳＰ１６とホスト１２間でストリームとされたデータを通すのに使用される。

【００５２】メッセージ

各タスクはメッセージを処理するメッセージハンドラーへのエントリーポイントを有する。メッセージはユーザ定義であり、タスク機能のための初期化パラメータおよびタスクをコントロールするコマンドを含む。タスクはその生成時に指定されるコールバックを介してホスト１２へメッセージを送る。タスクメッセージハンドラーのプロトタイプおよびホストコールバックのプロトタイプをここに示す。

【数４４】

*void TaskMsgHandler(T_ReplyRef replyref, T_MsgID MsgID, T_Count count, T_DSP_Word *buf);*
*void HostCallBack(T_ReplyRef replyref, T_MsgID MsgID, T_Count count, T_DSP_Word *buf);*

【００５３】ｒｅｐｌｙｒｅｆパラメータは送信者への返答を返送するのに使用されるインプルメンテーション依存基準値である。各Ｓｅｎｄ＿Ｍｅｓｓａｇｅコールについて、受信者はｒｅｐｌｙｒｅｆパラメータを使用してＲｅｐｌｙ＿Ｍｅｓｓａｇｅを呼び出さなければならない。実際のメッセージは次のようである。

【数４５】

Sent message:	MsgPktFlag	taskid	replyref	msgid	count	buf[.....]
Reply message:	MsgPktFlag	-1	replyref	msgid	count	buf[.....]

マルチワードデータは最下位語が最初に送られる。

【００５４】Ｓｅｎｄ＿Ｍｅｓｓａｇｅ関数内の０のＴａｓｋＩＤはシステムレベルメッセージを示す。システムレベルメッセージはＤＳＰ－ＡＰＩ関数を実施するのに使用される。

【００５５】メッセージ関数を下記に示す。

【数４６】

Send_Message

*BOOL Send_Message(T_TaskID TaskID, T_MsgID MsgID, T_Count count, T_DSP_Word *msgbuf, T_DSP_Word *replybuf, T_Size replybufsize, T_Count replycount, U16 *errcode);*

この関数はＴａｓｋＩＤにより指定されるタスクへユーザ定義メッセージを送る。ＭｓｇＩＤはメッセージを定義し、ｍｓｇｂｕｆは実際のメッセージデータを含んでいる。メッセージサイズはｃｏｕｎｔＴ＿ＤＳＰ＿Ｗｏｒｄｓである。メッセージへの返答はｒｅｐｌｙｂｕｆパラメータ内に含まれ、それは呼び出し者により提供されるサイズｒｅｐｌｙｂｕｆｓｉｚｅのバッファーを指示する。それは特定のメッセージに対する返答を処理するのに十分なサイズでなければならない。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数４７】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

【００５６】

【数４８】

Reply_Message

*BOOL Reply_Message(T_ReplyRef replyref, T_Count count, T_DSP_Word *buf, U16 *errcode);*

この関数はメッセージへ返答するのに使用される。ｒｅｐｌｙｒｅｆはオリジナルメッセージの送信者へ返答を返送するのに使用される参照（ｒｅｆｅｒｅｎｃｅ）であり、特定インプルメンテーション向けである。返答はｂｕｆパラメータ内に含まれそのサイズはＴ＿ＤＳＰ＿Ｗｏｒｄｓである。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数４９】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BAD_REPLY_REF

【００５７】チャネル

チャネルの概念は１つのプロセッサから別のプロセッサへ、あるいは同じプロセッサ上のタスク間でフレームベースデータを送信するのに使用される。生成されると、チャネルはデータを含むように指定された数およびサイズのフレームを割り当てる。最初に、チャネルは空フレームのリストを含んでいる。データを生じるタスクはデータを書き込む空フレームを要求し、書き込まれるとフレームはチャネルへ戻される。データを消費するタスクはチャネルから完全なフレームを要求し、空になるとフレームはチャネルへ戻される。フレームバッファーのこの要求および戻しにより、最小限のコピーでデータを動かすことができる。

【００５８】各タスクは指定された入力および出力チャネルを有する。チャネルが生成されると、それは１つのタスクへの入力、およびもう１つのタスクへの出力として指示される。チャネルのＩＤにはデバイスＩＤが含まれ、チャネルはプロセッサ間でデータを通すことができる。ホスト－ＤＳＰインターフェイスを横切るチャネルデータフローは下記のようである。

【数５０】

ChanPktFlag	Channel ID	Count	Data[...]

チャネル関数を下記に示す。

【００５９】

Create_Channel

*BOOL Create_Channel(T_DeviceID DevID, T_Size framesize, T_Count numframes, T_ChanID *ChannelID, U16 *errcode);*

この関数はデータフレームーベース通信チャネルを生成する。それはカウントおよびサイズが、それぞれ、ｎｕｍｆｒａｍｅｓおよびｆｒａｍｅｓｉｚｅ内に指定されている１組のフレームバッファーのコントロールを維持するチャネルコントロール構造を生成する。生成されると、チャネルはデータフレームを割り当てそれらをその空フレームリストへ加える。ＣｈａｎｎｅｌＩＤは新しいチャネルのＩＤを戻す。ＤｅｖＩＤが呼出プロセッサのものでなければ、呼出プロセッサとＤｅｖＩＤプロセッサの両方でチャネルコントロール構造が生成され、通信インターフェイスを横切るデータフローがコントロールされる。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数５１】

【数５２】

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_ALLOCATION_ERROR

【００６０】

【数５３】

Delete_Channel

*BOOL Delete_Channel(T_ChanID ChannelID, U16 *errcode);*

この関数はＣｈａｎｎｅｌＩＤにより指定される既存のチャネルを削除する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【００６１】

【数５５】

【数５４】

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_CHANNEL_NOT_FOUND

Request_Empty_Frame

*BOOL Request_Empty_Frame(T_LocalChanID Chn, T_DSP_Word **bufptr, BOOL WaitFlag, U16 *errcode);*

この関数は指定されたローカンチャネルＩＤから空フレームを要求する。ＣｈｎがＮＵＬＬであれば、タスクの出力チャネルが使用される。戻る時に、ｂｕｆｐｔｒはフレームバッファーへのポインターを含んでいる。ＷａｉｔＦｌａｇがＴＲＵＥであり、かつ利用可能なフレームバッファーがなければ、呼出者はバッファーを利用できるようになるまで保留される。ＷａｉｔＦｌａｇがＦＡＬＳＥであれば、機能はとにかく戻る。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数５６】

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE

【００６２】

【数５７】

Return_Full_Frame

*BOOL Return_Full_Frame(T_LocalChanID Chn, T_DSP_Word *bufptr, U16 *errcode);*

一度タスクがフレームバッファーを満たすと、この関数を使ってチャネルへ戻される。ｂｕｆｐｔｒにより指示されるバッファーは指定されたチャネルＩＤへ戻される。ＣｈｎがＮＵＬＬであれば、タスクの出力チャネルが使用される。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数５８】

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR

Request_Full_Frame

*BOOL Request_Full_Frame(T_LocalChanID Chn, T_DSP_Word **bufptr, BOOL WaitFlag, U16 *errcode);*

この関数は指定されたローカンチャネルＩＤからデータの全フレームを要求する。ＣｈｎがＮＵＬＬであれば、タスクの入力チャネルが使用される。戻る時に、ｂｕｆｐｔｒパラメータはフレームバッファーへのポインターを含んでいる。ＷａｉｔＦｌａｇがＴＲＵＥであり、かつ利用可能な完全なフレームバッファーがなければ、呼び出し者はバッファーが利用できるようになるまで保留される。ＷａｉｔＦｌａｇがＦＡＬＳＥであれば、関数はとにかく戻る。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数６０】

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE

【００６４】

【数６１】

Return_Empty_Frame

*BOOL Return_Empty_Frame(T_LocalChanID Chn, T_DSP_Word *bufptr, U16 *errcode);*

タスクはフレームバッファーからのデータを使用したら、この関数を使用してバッファーをチャネルへ戻さなければならない。ｂｕｆｐｔｒにより指示されるバッファーは指定されたチャネルＩＤへ戻される。ＣｈｎがＮＵＬＬであれば、タスクの入力チャネルが使用される。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数６２】

CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR

【００６５】

【数６３】

Set_Task_Input_Channel

*BOOL Set_Task_Input_Channel(T_Task *TaskID, T_ChanID ChanID, U16 *errcode);*

この関数はタスクの入力チャネルを指定されたチャネルＩＤに設定する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【００６６】

【数６４】

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND

【数６５】

Set_Task_Output_Channel

*BOOL Set_Task_Output_Channel(T_Task *TaskID, T_ChanID ChanID, U16 *errcode);*

この関数はタスクの出力チャネルを指定されたチャネルＩＤに設定する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数６６】

CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND

【００６７】ストリーム

ストリームはフレームに押し入ることはできないが、タスクに対して連続的に流出入するデータのために使用される。ストリームはヘッドおよび流出入時にデータを追跡するテールポインターを付随するサーキュラーバッファー（ＦＩＦＯ）からなっている。各タスクは指定された入力および出力ストリームを有することができる。ホスト－ＤＳＰインターフェイスを横切るストリームデータフローは次のようである。

【数６７】

StrmPktFlag	Stream ID	Count	Data[...]

ストリーム関数を下記に示す。

【数６８】

【００６８】

Create_Stream

*BOOL Create_Stream(T_DeviceID DevID, T_Size FIFOsize, T_StrmID *StreamID, U16 *errcode);*

この関数はＦＩＦＯベース通信ストリームを生成する。それはサイズＦＩＦＯｓｉｚｅのＦＩＦＯのコントロールを維持するストリームコントロール構造を生成する。生成されると、ストリームは空ＦＩＦＯを割り当て、ストリームに流出入するデータフローを処理するようにヘッドおよびテールポインターを初期化する。ＳｔｒｅａｍＩＤは新しいストリームのＩＤを戻す。ＤｅｖＩＤが呼出プロセッサのものでなければ、呼出プロセッサおよびＤｅｖＩＤプロセッサの両方にストリームコントロール構造が生成され、通信インターフェイスを横切って流れるデータをコントロールする。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数６９】

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_ALLOCATION_ERROR

【００６９】

【数７０】

Delete_Channel

*BOOL Delete_Stream(T_StrmID StreamID, U16 *errcode);*

この関数はＳｔｒｅａｍＩＤにより指定される既存のストリームを削除する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【００７０】

【数７２】

【数７１】

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND

Get_Stream_Count

*BOOL Get_Stream_Count(T_LocalStrmID StrmID, T_Count *count, U16 *errcode);*

この関数は現在ＳｔｒｍＩＤにより指定されるストリームＦＩＦＯ内にあるＴＤＳＰ__Ｗｏｒｄｓのカウントを要求する。ｃｏｕｎｔパラメータは戻った時に数を含んでいる。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数７３】

STRM_SUCCESS
STRM_STREAM_NOT_FOUND

【００７１】

【数７４】

Write_Stream

*BOOL Write_Stream(T_LocalStrmID Strm, T_DSP_Word bufptr, T_Count count, T_Count countwritten, U16 *errcode);*

この関数はＳｔｒｍにより指定されたストリームへＴＤＳＰ__Ｗｏｒｄｓのｃｏｕｎｔ数を書き込む。ＳｔｒｍがＮＵＬＬであれば、タスクの出力ストリームが使用される。データはｂｕｆｐｔｒパラメータにより指示される。戻った時に、ｃｏｕｎｔｅｒｗｒｉｔｔｅｎは実際に書き込まれるＴ__ＤＳＰ__Ｗｏｒｄｓ数を含んでい

る。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【数７５】

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND
STRM_STREAM_OVERFLOW

Read_Stream

*BOOL Read_Stream(T_LocalStrmID Strm, T_DSP_Word *bufptr, T_Count maxcount, BOOL WaitFlag, T_Count *countread, U16 *errcode);*

この関数はＳｔｒｍにより指定されたストリームからデータを読み出す。ＳｔｒｍがＮＵＬＬであれば、タスクの入力ストリームが使用される。データはｂｕｆｐｔｒにより指定されたバッファー内に格納される。ｍａｘｃｏｕｎｔまでＴＤＳＰ＿Ｗｏｒｄｓがストリームから読み出される。ｃｏｕｎｔｅｒｅａｄパラメータは読み出したデータの実際のカウントを含んでいる。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【００７２】

【数７６】

【数７７】

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND

【００７３】

【数７８】

Set_Task_Input_Stream

*BOOL Set_Task_Input_Stream(T_Task *TaskID, T_StrmID StrmID, U16 *errcode);*

この関数は指定されたストリームＩＤへタスクの入力ストリームを設定する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【００７４】

【数８０】

【数７９】

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND

Set_Task_Output_Stream

*BOOL Set_Task_Output_Stream(T_Task *TaskID, T_StrmID StrmID, U16 *errcode);*

この関数は指定されたストリームＩＤへタスクの出力ストリームを設定する。ｅｒｒｃｏｄｅパラメータは下記の考えられる結果を含む。

【００７５】

【表３】

【数８１】

STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND

表８　データタイプ
ここで使用するデータタイプを表８に示す。

シンボル	説明
S8	Signed 8-bit integer. （署名８ビット整数）
U8	Unsigned 8-bit inleger. （非署名８ビット整数）
S16	Signed 16-bit integer. （署名１６ビット整数）
U16	Unsigned 16-bit integer. （非署名１６ビット整数）
S32	Signed 32-bit integer. （署名３２ビット整数）
U32	Unsigned 32-bit integer. （非署名３２ビット整数）
T_HostWord	A word on the Host processor. （ホストプロセッサ上の語）
T_DSP_Word	A word on the DSP processor. （ＤＳＰプロセッサ上の語）
BOOL	Boolean value(TRUE or FALSE). （ブール値(TRUE 又は FALSE）
T_HostPtr	Pointer on the Host processor. （ホストプロセッサ上のポインター）
T_DSP_Ptr	Pointer on the DSP processor. （ＤＳＰプロセッサ上のポインター）
T_DeviceID	Processor device ID. （プロセッサデバイスＩＤ）
T_TaskID	A structure containing fields for a device ID and a processor local task ID. （デバイスＩＤ及びプロセッサローカルタスクＩＤ用フィールドを含む構造）
T_ChanID	A structore containing fields for a device ID and a processor local channel ID. （デバイスＩＤ及びプロセッサローカルチャネルＩＤ用フィールドを含む構造）
T_MsgID	Message ID. （メッセージＩＤ）
T_DSP_ID	An object ID on the DSP. （ＤＳＰ上のオブジェクトＩＤ）
T_Count	Data type for a count. （カウントのデータタイプ）
T_Size	Data type for a size. （サイズのデータタイプ）
T_HostCallBack	Value used when tasks send message back to the Host. （タスクがホストへメッセージを返送する時の値）
T_ReplyRef	Message reply reference. （メッセージ返答参照）
T_LocalTaskID	Local task ID. （ローカルタスクＩＤ）
T_LocalChanID	Local channel ID. （ローカルチャネルＩＤ）
T_DSP_Name	Name for DSP objects(RTOS dependent). （ＤＳＰオブジェクト名（ＲＴＯＳ依存））
T_CodeHdr	Code header structure for a DSP Library entry. （ＤＳＰライブラリエントリ用コードヘッダー構造）
T_TaskCreate	Task creation structure. （タスク生成構造）

【００７６】システムメッセージ
これらの表はデバイス間を通過するメッセージを定義する（すなわち、ホストからＤＳＰ１６へ）。メッセージをデバイスへ実際にルーティングするのに使用されるため、対応する関数呼出し内にパラメータとして存在するデバイスＩＤはメッセージには内蔵されない。同様に、関数呼出しの上半部としてデバイスＩＤを含むタスクＩＤはメッセージ内にデバイスＩＤを含まず、ＤＳＰのロードタスクＩＤ部しか含んでいない。
【００７７】
【表４】

表４ ＤＳＰ－ＡＰＩ メッセージ

メッセージ	パラメータ送出	パラメータ返答	方向 Host ⇔DSP
GET_MIPS	None	U32 mips	→
GET_MEM_AVAIL		T_Size progmem T_Size datamem	→
ALLOC_MEM	U16 mempage T_Size size	T_DSP_Word *memptr U16 errcode	→
FREE_MEM	U16 mempage T_DSP_Word *memptr	U16 errcode	→
PUT_BLOB	T_DSP_Ptr destaddr U16 mempage T_Size size T_DSP_Word BLOB[size]	U16 errcode	→
CREATE_TASK	T_TaskCreate tcs	T_TaskID TaskID U16 errcode	→
START_TASK	T_TaskID TaskID	U16 errcode	→
SUSPEND_TASK	T_TaskID TaskID	U16 errcode	→
RESUME_TASK	T_TaskID TaskID	U16 errcode	→
DELETE_TASK	T_TaskID TaskID	U16 errcode	→
CHANGE_PRIORITY	T_TaskID TaskID U16 newpriority	U16 oldpriority U16 errcode	→
GET_TASK_STATUS	T_TaskID TaskID	U16 status U16 priority T_ChanID Input T_ChanID Output U16 errcode	→
GET_ID	T_DSP_Name Name	T_DSP_ID ID U16 errcode	→

【００７８】

【表５】

表５ インターフェイスレイヤ／チャネルインターフェイスレイヤメッセージ

メッセージ	パラメータ送出	パラメータ返答	方向 Host ⇔DSP
CREATE_CHANNEL	T_Size framesize T_Count numframes	T_ChanID ChannelID U16 errcode	→
DELETE_CHANNEL	T_ChanID ChannelID	U16 errcode	→
CREATE_STREAM	T_Size FIFOsize	T_StrmID StreamID U16 errcode	→
DELETE_STREAM	T_StrmID StreamID	U16 errcode	→

【００７９】

【表６】

表６ デバッグメッセージ

メッセージ	パラメータ送出	パラメータ返答	方向 Host ⇔DSP
READ_MEM	U16 mempage T_DSP_Ptr addr T_Count count	T_DSP_Word mem[count] U16 errcode	→
WRITE_MEM	U16 mempage T_DSP_Ptr addr	U16 errcode	→
	T_Count count T_DSP_Word mem[count]		
READ_REG	U16 RegID	DSP_WORD regvalue U16 errcode	→
WRITE_REG	U16 RegID T_DSP_Word regvalue	U16 errcode	→
SET_BREAK	T_DSP_Ptr addr	U16 errcode	→
CLR_BREAK	T_DSP_Ptr addr	U16 errcode	→
BREAK_HIT	T_DSP_Ptr addr	U16 ACK	←

【００８０】ダウンローディングネーティブコード

図４－図６はネーティブコードをターゲットプロセッサ（すなわち、ホスト１２もしくはＤＳＰ１６）へ確実かつ効率的にダウンローディングする実施例を示す。コードをダウンローディングするこの実施例は、例えば、インターネット、もしくは他のグローバルネットワーク、ローカルもしくはワイドエリアネットワーク、もしくはＰＣカードやスマートカード等の周辺装置からコードをダウンローディングするのに使用できる。

【００８１】図４にＪＡＶＡ Ｂｅａｎ９０の実施例を示し、Ｂｅａｎ９０はネーティブコード９２のラッパー（ｗｒａｐｐｅｒ）として作用する。Ｂｅａｎはさらに Ｃｏｄｅ Ｔｙｐｅ属性９４ａ、Ｃｏｄｅ Ｓｉｚｅ属性９４ｂおよびＭＩＳＰ要求属性９４ｃとして表記されているいくつかの属性９４を含んでいる。Ｂｅａｎ９０はＬｏａｄ Ｃｏｄｅアクション９６ａ、Ｌｏａｄ Ｐａｒａｍｅｔｅｒｓアクション９６ｂおよびＥｘｗｃｕｔｅ Ｐａｒａｍｅｔｅｒ９６ｃを含むいくつかのアクション９６を有している。

【００８２】動作において、Ｌｏａｄ Ｃｏｄｅアクション９６ａは外部ネーティブコード（ターゲットプロセッサにとってネーティブ）をＢｅａｎ内へロードするの

に使用される。ＪＡＶＡ　Ｂｅａｎｓはパーシステンス（ｐｅｒｓｉｓｔｅｎｃｅ）を有するため、Ｂｅａｎ９０はネーティブコード９２および属性９４を含むその内部状態を格納することができる。Ｌｏａｄ　Ｐａｒａｍｅｔｅｒｓアクション９６ｂはネーティブコード９２から（例えば、前記したＣＯＦＦファイルフォーマットを使用して）パラメータを検索し属性９４ａ－ｃとして格納する。Ｅｘｅｃｕｔｅアクション９６ｃはＤＳＰ１６内にインストールされたタスクを実行する。

【００８３】Ｂｅａｎ９０を使用してコードをターゲットプロセッサへダウンロードする様子を図５に示す。この例では、ターゲットプロセッサはＤＳＰ１６（もしくは、多数のＤＳＰ１６の中の１つ）であるものとするが、それはネーティブコードをホストプロセッサ１２へダウンロードするのにも使用できる。さらに、所望のＢｅａｎ９０はＬＡＮサーバもしくはインターネットサーバ等のネットワークサーバ内に常駐しているものとするが、Ｂｅａｎはスマートカード等のプラットホーム１０と通信する任意のデバイス内に常駐することができる。ワイヤレスデータプラットホーム１０については、ネットワークサーバ１００への接続はしばしばワイヤレスである。

【００８４】図５において、プラットホーム１０はネットワークサーバ１００に接続されている。図２に詳細に示すホストプロセッサ１２はＪＡＶＡバーチャルマシン４４を介して１つ以上のＪＡＶＡアプレット４０を実行することができる。新しいコードをダウンロードするために、ホスト１２はネットワークサーバ１００からＢｅａｎ９０を含むアプレットをロードするか、もしくはアプレットを含まないＢｅａｎをサーバ１００からダウンロードすることができる。ラッパーＢｅａｎ９０が検索されると、ネーティブコードのサイズ、コードタイプ（コードはどのプロセッサ用か）および必要なＭＩＰＳを質問することができる。所期のプロセッサがコード９２を実行するのに十分な資源を有する場合には、コード９２は所期のプロセッサ、図５に示すアーキテクチュア内のホストプロセッサ１２もしくはＤＳＰ１６、で実行するようにインストールすることができる。典型的には、ネーティブコード９２は非リンク、コンパイルされたコードである。したがって、ＤＳＰ－ＡＰＩ５０のクロスリンカー８２はコードを利用可能なメモリ位置へリンクする。Ｂｅａｎは２進ネーティブコード９２をダイナミッククロスリンカー８２へ通し、それがコードをインストールして実行する。

【００８５】典型的に、ネーティブコードのダウンロードが生じるのは、その中にＤＳＰ関数が所望されるアプレットをユーザが実行している場合である。最初に、アプレットは所望のコードがタスク６０としてＤＳＰ内にインストールされている、またはＤＳＰライブラリ５８内で利用可能であるかをチェックする。もしそうならタスクはダウンロードなしで実行できる。

【００８６】タスクがＤＳＰ１６もしくはＤＳＰライブラリ５８内に格納されていなければ、オブジェクト（ここでは、“ＤＳＰＬｏａｄｅｒ”オブジェクトと言う）を生成してＢｅａｎ９０をロードすることができる。ＤＳＰＬｏａｄｅｒクラスがホスト１２上でローカルであれば、ＪＡＶＡはＢｅａｎもローカルに利用可能であるかをチェックする。最初の場合、ローカルに格納されたコードを有するＢｅａｎがあることがある。そうであれば、ＢｅａｎからのコードはＤＳＰ（もしくは、Ｃｏｄｅ　Ｔｙｐｅにより指定されるいずれかのプロセッサ）へインストールされる。コードのないＢｅａｎがローカルに格納される場合には、Ｂｅａｎは適切なサーバからコードを検索することができる。

【００８７】一方、ＤＳＰＬｏａｄｅｒオブジェクトがローカルではなければ、ＪＡＶＡはアプレットを書き込んだサーバからＢｅａｎ９０をロードする。次に、Ｂｅａｎからのコードは前記したようにインストールされる。

【００８８】ＪＡＶＡ　Ｂｅａｎの使用に関連してネーティブコードのダウンローディングが説明されるが、ＡｃｔｉｖｅＸアプレット等の他の言語内にコードを包み込んで達成することもできる。

【００８９】ＪＡＶＡ　Ｂｅａｎ（もしくは、他のアプレット）をネーティブコードのラッパーとして使用することは著しい利点がある。最初に、複数のプロセッサの中の１つへコードをローディングするのに単純な標準的方法でよい。Ｂｅａｎが生成され、コードがＢｅａｎへロードされて適切なプロセッサとリンクされる。Ｂｅａｎ内にコードをラッピングすることなく、プロセスは数百のステップをとることができる。第２に、多数のネーティブコード片を１つのアプレットにより結合し、１つのアプレットを使用する多数の個別のルーチンから複雑なアプリケーションを発生してルーチンを所望により結合することができる。第３に、言語のセキュリティ機能を利用することができ、Ｂｅａｎ９０内のＪＡＶＡコードだけでなくネーティブコード９２も保護される。ＡｃｔｉｖｅＸ等の他の言語もセキュリティ機能を有する。

【００９０】セキュリティ

２つの最重要セキュリティ機能はデジタル署名および暗号化である。ＪＡＶＡＢｅａｎもしくはＡｃｔｉｖｅＸアプレットはコードソースにより署名することができ、Ｂｅａｎもしくはアプレットがダウンロードされると、署名は信頼できるソースのリストを有する受信アプリケーションにより照合される。Ｂｅａｎもしくはアプレットに信頼できるソースの署名があれば、標準技術を使用して解読することができる。したがって、ネーティブコードは伝送中にＢｅａｎもしくはアプレットのコードと共に暗号化され、コードの非認可修正が防止される。ネーティブコードは安全であり信頼できるソースから来る

ため、属性も正確であるものと信頼することができる。

【００９１】図６はＪＡＶＡ　Ｂｅａｎを使用するプロセッサのネーティブコードダウンローディングプロセスを説明するフロー図であり、ネーティブコードは同様な技術を使用して異なる言語のアプレットで包むことができることがお判りであろう。ステップ１１０において、暗号化されたデジタル署名Ｂｅａｎ９０はＪＡＶＡバーチャルマシンを動かすデバイスへダウンロードされる。ステップ１１２において、署名が照合される。信頼できるソースとして表記されたソースからのものでなければ、ステップ１１４において例外処理がイネーブル（ｅｎａｂｌｅ）される。Ｂｅａｎが信頼できるソースから来るものである場合、ユーザがそのソースで不自由でなければ、例外処理関数はユーザにＢｅａｎを受け入れる機会を与えることができる。署名が無効であれば、例外処理はＢｅａｎ９０を削除してユーザへ適切なエラーメッセージを送ることができる。

【００９２】署名が有効で信頼できるソースから来るものであれば、ステップ１１６においてＢｅａｎが解読される。このステップではＪＡＶＡコードとＢｅａｎ内のネーティブコードの両方が解読される。ステップ１１８において、Ｂｅａｎ９０から属性が検索されステップ１２０において、アプレットは適切なプロセッサがコードを実行するのに十分な資源を有するかどうかを確認する。十分な資源を有しない場合には、例外処理ステップ１１４はネーティブコードのインストールを拒絶することができ、あるいは資源を解放するステップをとることができる。十分な資源がある場合には、ステップ１２２においてコードはクロスリンカーを使用してリンクされ所望のプロセッサにインストールされる。ステップ１２４において、ネーティブコードが実行される。

【００９３】Ｂｅａｎ９０のサンプルＪＡＶＡスクリプトを下記に示す。

【表７】

```
package ti.dsp.loader;

import java.awt.*;
import java.io.*;
import java.net.*;

public class NativeBean extends Canvas implements Serializable
{
    public NativeBean() {

        setBackground(Color.white);

        funcData = new ByteArrayOutputStream();

        try {
            funcCodeBase = new URL("http://localhost");
        }
        catch (MalformedURLException e) {
```

【表８】

```
    }
}

public Dimension getMinimumSize() {

    return new Dimension(50, 50);
}

public void loadCode() {

    URL baseURL = null;

    try {
        baseURL = new URL(funcCodeBase.toString() + "/" + myFunction);
    }
    catch (MalformedURLException e) {
    }

    DataInputStream source = null;
    int read;
    byte[] buffer;

    buffer = new byte[1024];
    try {
        source = new DataInputStream(baseURL.openStream());
    }
    catch (IOException e) {
        System.out.println("IOException creating streams: " + e);
    }

    codeSize = 0;

    funcData.reset();

    try {
        while (true) {

            read = source.read(buffer);

            if (read == -1)
                break;

            funcData.write(buffer, 0, read);
        }
    }
    catch (IOException e) {
        System.out.println("IOException: " + e);
    }
```

【表９】

```
        codeSize = funcData.size();
        System.out.println("Code size = " + codeSize);

        try {
            source.close();
        }
        catch (IOException e) {
            System.out.println("IOException closing: " + e);
        }
    }

    public synchronized String getFunctionName() {

        return myFunction;
    }

    public void setFunctionName(String function) {

        myFunction = function;
    }

    public synchronized String getCodeBase() {

        return funcCodeBase.toString();
    }

    public void setCodeBase(String newBase) {

        try {
            funcCodeBase = new URL(newBase);
        }
        catch (MalformedURLException e) {
        }
    }

    public void installCode() {

        FileOutputStream destination = null;
        File libFile = new File(myFunction);

        try {
            destination = new FileOutputStream(libFile);
        }
        catch (IOException e) {
            System.out.println("IOException creating streams: " + e);
        }

        if (destination != null) {
```

【表１０】

```
                try {
                    funcData.writeTo(destination);
                }
                catch (IOException e) {
                    System.out.println("IO Exception installing native code: " + e);
                }
            }
        }
            linkCode(funcData)

        public void loadParameters() {
        }

        public void execute() {
        }

        public synchronized int getCodeSize() {

            return codeSize;
        }

        public synchronized int getCodeType() {

            return codeType;
        }

        public void setCodeType(int newType) {

            codeType = newType;
        }

        private int codeSize = 0;
        private int codeType = 1;
        private String myFunction = "";
        private URL funcCodeBase = null;
        private ByteArrayOutputStream funcData = null;
}
```

【００９４】前記したスクリプトにおいて、ＮａｔｉｖｅＢｅａｎ（）ルーチンはネーティブコードを保持するＢｅａｎ９０を生成する。ｌｏａｄＣｏｄｅ（）ルーチンはサーバからネーティブコードを得る。ｇｅｔＦｕｎｃｔｉｏｎａｌＮａｍｅ（）およびｇｅｔＣｏｄｅＢａｓｅ（）ルーチンは属性を検索する。ｉｎｓｔａｌｌＣｏｄｅ（）ルーチンはクロスリンカーを呼び出してネーティブコードをＤＳＰにリンクしリンクしたコードをロードする。ｌｏａｄＰａｒａｍｅｔｅｒｓ（）ルーチンはネーティブコードを調べてその属性を確認するようＢｅａｎを命令する。ｇｅｔＣｏｄｅｓｉｚｅ（）およびｇｅｔＣｏｄｅｔｙｐｅ（）ルーチンは属性を要求するアプレットへ転送する。

【００９５】本発明の詳細説明はある代表的な実施例に向けられたが、当業者ならば別の実施例だけでなくこれらの実施例のさまざまな修正が考えられるであろう。請求の範囲に入る修正および別の実施例は全て本発明に含まれるものとする。

【００９６】以上の説明に関して更に以下の項を開示する。

（１）移動電子装置であって、ネィティブコードを実行するコプロセッサと、ホストプロセッサシステムおよびプロセッサ独立コードに対応するネィティブコードを実行するように作動するホストプロセッサシステムであって、デジタル信号コプロセッサにより実施されるタスクをダイナミックに変化させるように作動するホストプロセッサシステムと、前記ホストプロセッサシステムと前記コプロセッサ間の通信回路と、を含む移動電子装置。

【００９７】（２）第１項記載の移動電子装置であって、前記コプロセッサはデジタル信号処理装置である、移動電子装置。

【００９８】（３）前記いずれか１項記載の移動電子装

置であって、前記プロセッサ独立コードはＪＡＶＡを含む、移動電子装置。

【００９９】（４）前記いずれか１項記載の移動電子装置であって、前記ホストプロセッサシステムは前記コプロセッサのネーティブコードを発生することができる、移動電子装置。

【０１００】（５）前記いずれか１項記載の移動電子装置であって、前記ホストプロセッサシステムはプロセッサ独立ソースコードをコンパイルすることにより前記コプロセッサのネーティブコードを発生することができる、移動電子装置。

【０１０１】（６）前記いずれか１項記載の移動電子装置であって、前記ホストプロセッサシステムはソースコードの識別されたブロックをコンパイルする、移動電子装置。

【０１０２】（７）前記いずれか１項記載の移動電子装置であって、前記ホストプロセッサシステムはコプロセッサで実行できるソースコードのブロックを識別してコードの前記ブロックをコンパイルする、移動電子装置。

【０１０３】（８）前記いずれか１項記載の移動電子装置であって、さらに前記コプロセッサへダウンロードして実行することができるルーチンのライブラリを格納するメモリを含む、移動電子装置。

【０１０４】（９）前記いずれか１項記載の移動電子装置であって、さらにハードウェア言語アクセラレータを含む、移動電子装置。

【０１０５】（１０）前記いずれか１項記載の移動電子装置であって、前記ハードウェアアクセラレータはＪＡＶＡアクセラレータを含む、移動電子装置。

【０１０６】（１１）第１項記載の移動電子装置であって、さらにネットワークからデータを受信するネットワークインターフェイス回路を含む、移動電子装置。

【０１０７】（１２）移動電子装置の制御方法であって、コプロセッサにおいてネィティブコードを実行するステップと、ホストプロセッサシステムにおいてネィティブコードおよびプロセッサ独立コードを実行するステップと、デジタル信号コプロセッサにより実施されるタスクを前記ホストプロセッサシステムによりダイナミックに変えるステップと、前記ホストプロセッサシステムと前記コプロセッサ間で通信を行うステップと、を含む方法。

【０１０８】（１３）第１２項記載の方法であって、コプロセッサにおいてネィティブコードを実行する前記ステップは、デジタル信号処理装置においてネーティブコードを実行するステップを含む、方法。

【０１０９】（１４）第１２項、第１３項記載の方法であって、さらに前記一般的処理システムにおいて前記コプロセッサのネィティブコードを発生するステップを含む、方法。

【０１１０】（１５）第１４項記載の方法であって、ネィティブコードを発生する前記ステップはプロセッサ独立ソースコードをコンパイリングしてネーティブコードを発生するステップを含む、方法。

【０１１１】（１６）第１５項記載の方法であって、さらに前記ソースコードのブロックを識別して前記コプロセッサで実行するようコンパイルするステップを含む、方法。

【０１１２】（１７）第１２項－第１６項記載の方法であって、さらに前記ホストプロセッサシステムから前記コプロセッサへダウンロードして実行するルーチンのライブラリを格納するステップを含む、方法。

【０１１３】（１８）移動電子装置であって、複数個のコプロセッサと、ホストプロセッサシステムであって、ソースコードを実行し、前記１個以上のコプロセッサで実行されるソースコードの１つ以上のセクションを識別し、ソースコードの識別した各セクションについて、対応するコプロセッサを決定し、ソースコードの識別した各セクションについて、コードの前記識別したセクションを前記対応するコプロセッサに関連するネーティブコードへコンパイルして前記対応するコプロセッサへインストールする、ように作動する前記ホストプロセッサシステムと、前記ホストプロセッサシステムと前記コプロセッサ間の通信回路と、を含む、移動電子装置。

【０１１４】（１９）移動電子装置の制御方法であって、ホストプロセッサシステムでソースコードを実行するステップと、１個以上のコプロセッサで実行されるソースコードの１つ以上のセクションを識別するステップと、ソースコードの識別した各セクションについて、対応するコプロセッサを決定するステップと、ソースコードの識別した各セクションについて、コードの前記識別したセクションを前記対応するコプロセッサに関連するネーティブコードへコンパイリングして前記ネーティブコードを前記対応するコプロセッサへインストールするステップと、前記ホストプロセッサシステムと前記コプロセッサ間の通信を行うステップと、を含む、方法。

【０１１５】（２０）複数個のプロセッサ（１２，１６）を含むワイヤレスデータプラットホーム（１０）。タスクが実施される時に情報を通信できるようにプロセッサ間に通信チャネルが開設される。１個のプロセッサで実施されるダイナミッククロスコンパイラー（８０）がコードを別のプロセッサのためのネーティブ処理コードへコンパイルする。ダイナミッククロスリンカー（８２）が他のプロセッサのためにコンパイルされたコードをリンクする。ネーティブコードはそれを包むＪＡＶＡ　Ｂｅａｎ（９０）（もしくは他の言語タイプ）を使用してプラットホームへダウンロードすることもできる。ＪＡＶＡ　Ｂｅａｎはセキュリティのために暗号化しデジタル署名することができる。

【０１１６】関連出願の相互参照

本出願は同じ日付で出願され本開示の一部としてここに

組み入れられている、ウースリー等の米国特許出願第０８／９９５，６００号“Ｍｏｂｉｌｅ　Ｃｏｍｍｕｎｉｃａｔｉｏｎ　Ｓｙｓｔｅｍ　ｗｉｔｈ　Ｃｒｏｓｓ　Ｃｏｍｐｉｌｅｒ　ａｎｄ　Ｃｒｏｓｓ　Ｌｉｎｋｅｒ”（アットニードケット第２６４５３号）、ブリューワーの米国特許出願第０８／９９５，５９７号“Ｍｅｔｈｏｄ　ａｎｄ　Ａｐｐａｒａｔｕｓ　ｆｏｒ　Ｐｒｏｖｉｄｉｎｇ　Ｄｏｗｎｌｏａｄａｂｌｅ　Ｆｕｎｃｔｉｏｎａｌｉｔｙ　ｔｏ　ａｎ　Ｅｍｂｅｄｄｅｄ　Ｃｏｐｒｏｃｅｓｓｏｒ”（アットニードケット第２６４４０号）、およびブリューワーの米国特許出願第０８／９９５，６０３号“Ｍｅｔｈｏｄ　ａｎｄ　Ａｐｐａｒａｔｕｓ　ｆｏｒ　Ｅｘｔｅｎｄｉｎｇ　Ｓｅｃｕｒｉｔｙ　Ｍｏｄｅｌ　ｔｏ　Ｎａｔｉｖｅ　Ｃｏｄｅ”（アットニードケット第２６４３９号）に関連している。

【図面の簡単な説明】

【図１】一般的なワイヤレスデータ処理に特に適したプラットホームアーキテクチュアのブロック図。

【図２】図１のプラットホームの機能ブロック図。

【図３】ダイナミッククロスコンパイリングおよびダイナミッククロスリンキング機能の機能ブロック図。

【図４】デバイスへダウンロードするＪＡＶＡ　Ｂｅａｎラッパー内に包まれたプロセッサで実行するネーティブコードの実施例。

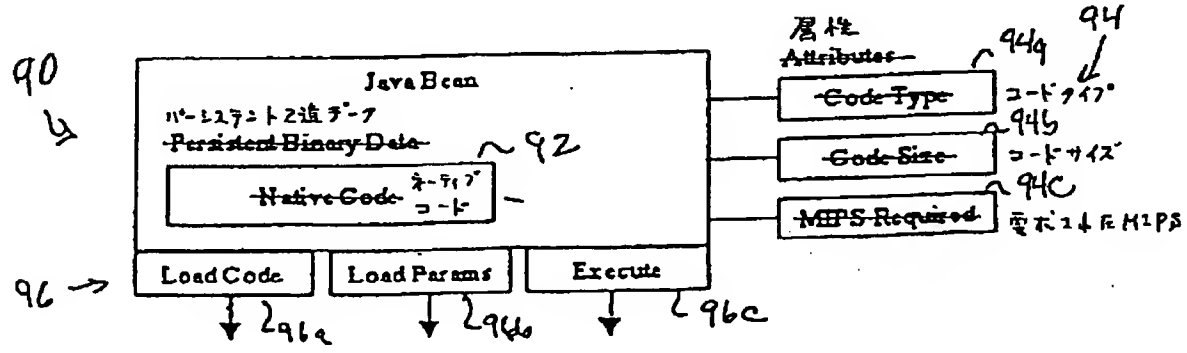
【図５】遠隔サーバ上に配置されたＪＡＶＡ　Ｂｅａｎからデバイス上のプロセッサへ包まれたネーティブコードを転送する操作を示す図。

【図６】図５の操作に関連するセキュリティ機能を記述するフロー図。

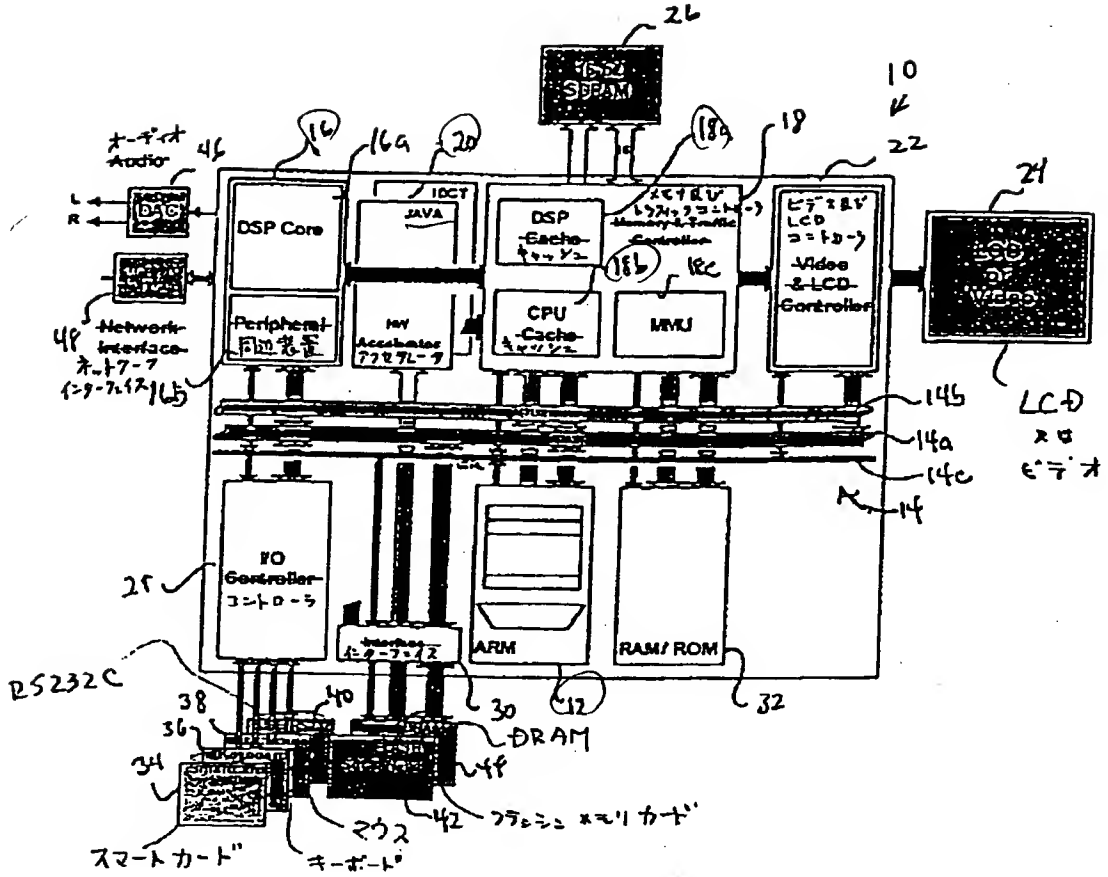
【符号の説明】

１０　ワイヤレスデータプラットホーム
１２　汎用ホストプロセッサ
１４　バス構造
１６　ＤＰＳ（デジタル信号処理装置）
１８　トラフィックコントローラ
２０　ハードウェアアクセラレータ回路
２２　ビデオおよびＬＣＤコントローラ
２４　ＬＣＤもしくはビデオディスプレイ
２６　主記憶装置
２８　Ｉ／Ｏコントローラ
３０　インターフェイス
３２　ＲＡＭ／ＲＯＭ
３４　スマートカード
３６　キーボード
３８　マウス
４０　シリアルポート
４２　フラッシュメモリカード
４４　ＤＲＡＭカード
４６　ＤＡＣ（デジタル／アナログコンバータ）
４８　ネットワークインターフェイス
５０　ＤＳＰ　ＡＰＩ（アプリケーションプログラムインターフェイス）
５２　ホストＤＳＰインターフェイスレイヤ
５４　ＤＳＰデバイスドライバ
５６　ホストＲＴＯＳ（リアルタイムオペレーティングシステム）
５８　ＤＳＰライブラリ
６０　タスク
６２　ホスト－ＤＳＰインターフェイスレイヤ
６４　ＤＲＴＯＳ
６６　ホストデバイスドライバ
７０　ホスト－ＤＳＰインターフェイス
８０　ダイナミッククロスコンパイラー
８２　ダイナミッククロスリンカー
９０　ＪＡＶＡ　Ｂｅａｎ
９２　ネーティブコード
９４　属性
９６　アクション
１００　ネットワークサーバ

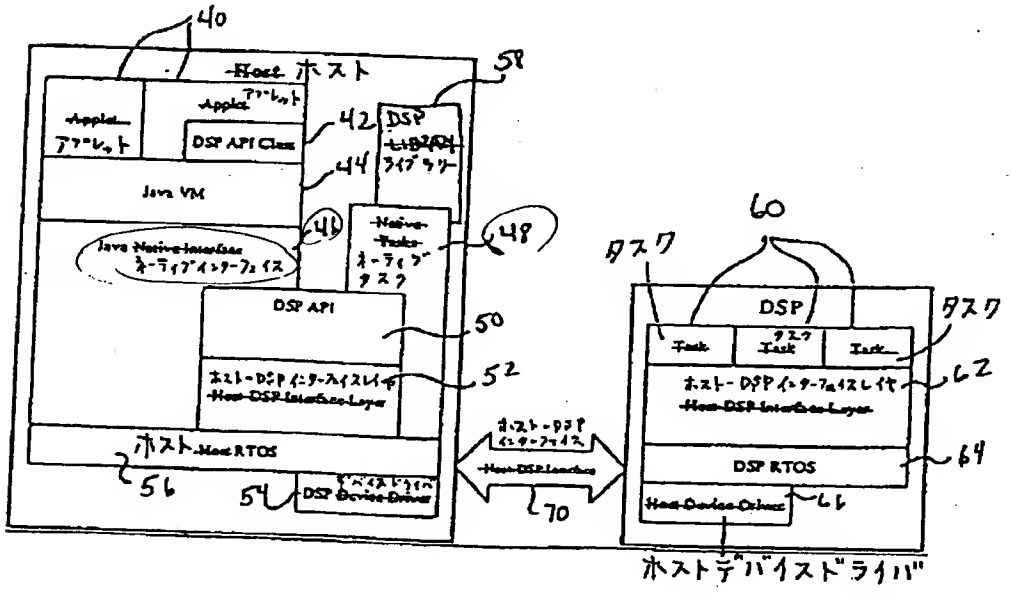
【図４】



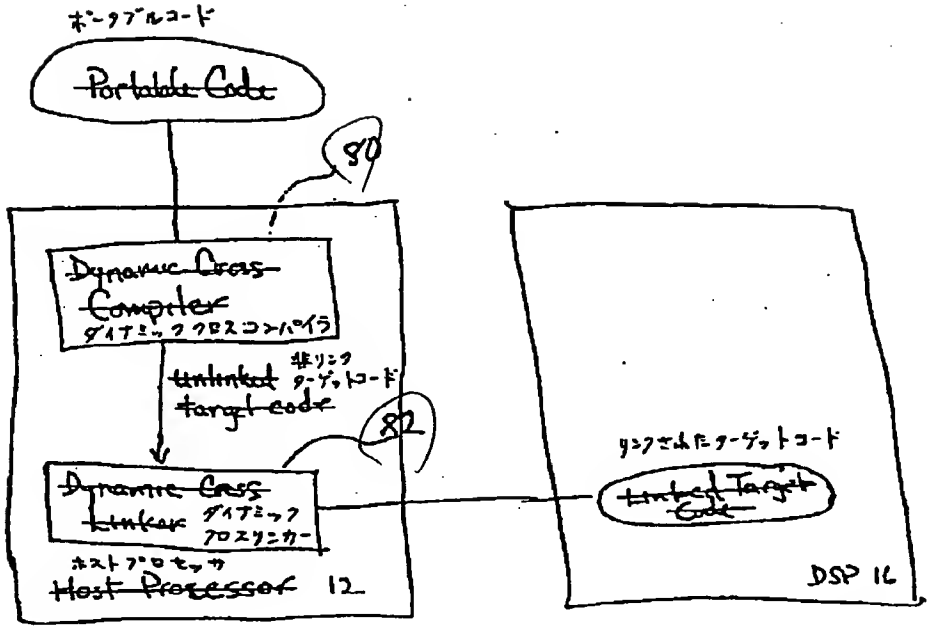
【図１】


26
SRAM
10
オーディオ
Audio
46
16
16a
20
18a
18
22
24
DAC
L
R
DSP Core
JAVA
DSP
Cache
キャッシュ
メモリ及びトランスファコントローラ
Memory & Traffic Controller
ビデオ及びLCDコントローラ
Video & LCD Controller
LCD or Video
Peripheral
周辺装置
MMU
アクセラレータ
CPU
Cache
キャッシュ
MMU
48
Network Interface
ネットワークインターフェイス
16b
16c
14b
14a
14c
14
LCD
又は
ビデオ
I/O
Controller
コントローラ
25
Interface
インターフェイス
ARM
RAM/ROM
RS232C
38
40
36
30
12
32
DRAM
34
44
42
フラッシュメモリカード
マウス
スマートカード
キーボード

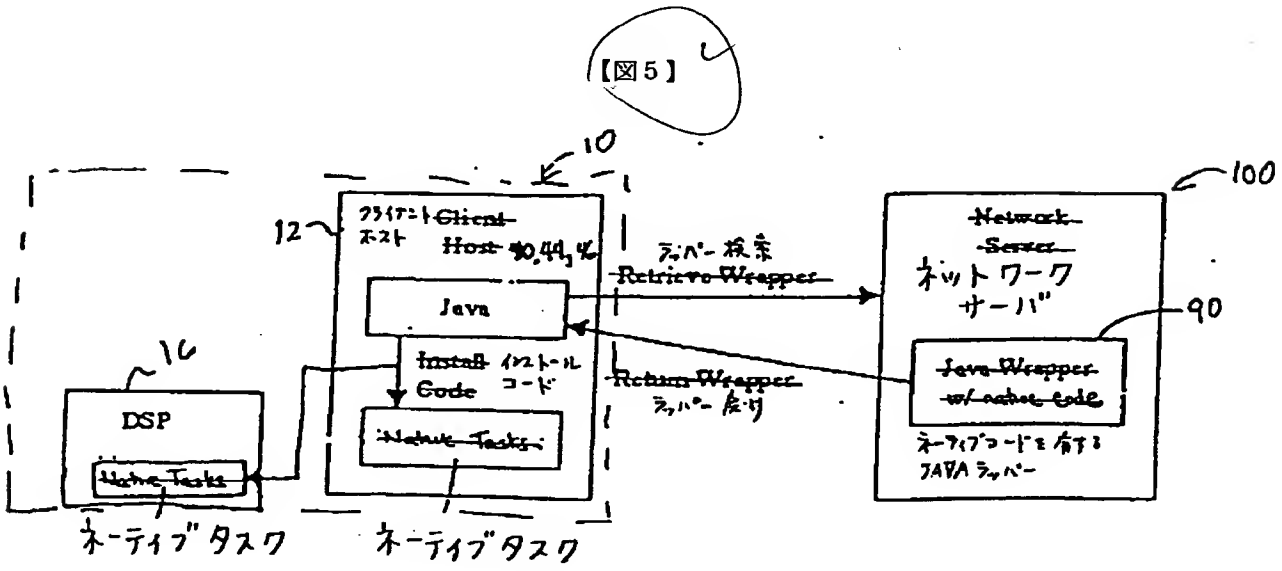
【図２】


40
Host ホスト
58
Applet
アプレット
Applet
アプレット
DSP API Class
42
DSP
Library
ライブラリ
44
Java VM
46
Java Native Interface
ネーティブインターフェイス
Native Tasks
ネーティブタスク
48
DSP API
50
ホスト-DSPインターフェイスレイヤ
Host-DSP Interface Layer
52
ホスト Host RTOS
56
54
DSP Device Driver
ホスト-DSPインターフェイス
Host-DSP Interface
70
タスク
60
DSP
Task
タスク
Task
Task
ホスト-DSPインターフェイスレイヤ
Host-DSP Interface Layer
62
DSP RTOS
64
Host Device Driver
66
ホストデバイスドライバ

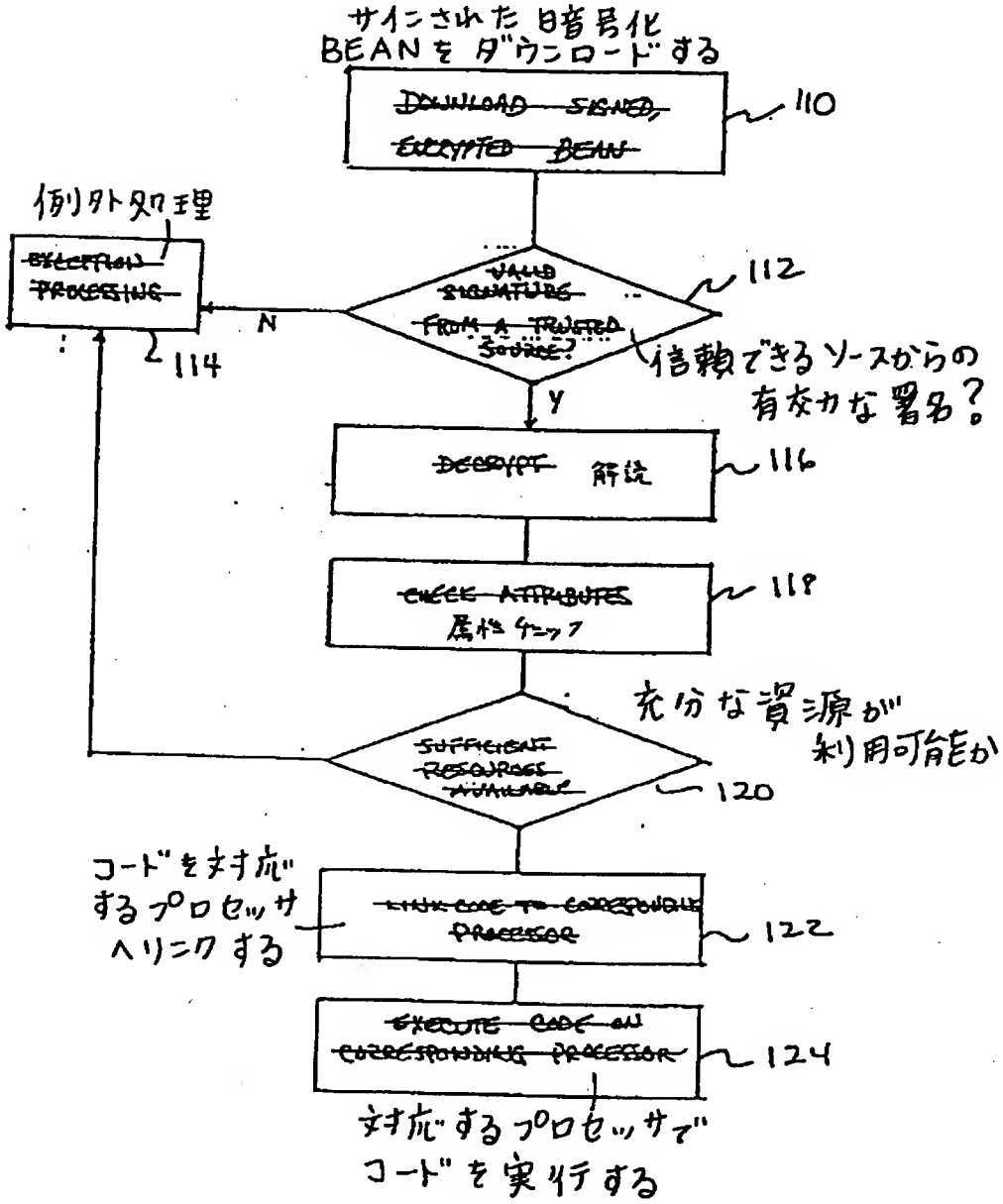
【図３】


ポータブルコード
Portable Code
80
Dynamic Cross Compiler
ダイナミック クロスコンパイラ
unlinked 非リンク
target code ターゲットコード
82
Dynamic Cross Linker ダイナミック クロスリンカー
ホストプロセッサ
Host Processor 12
リンクされたターゲットコード
Linked Target Code
DSP 16

【図５】


10
100
12
クライアント Client
ホスト Host
Java
ラッパー検索
Retrieve Wrapper
Network Server
ネットワーク サーバ
90
Install インストール Code コード
Return Wrapper ラッパー戻り
Java Wrapper w/ native code
ネーティブコードを有する JAVA ラッパー
16
DSP
Native Tasks
ネーティブタスク
Native Tasks
ネーティブタスク

【図6】



フロントページの続き

(72)発明者 マシュー エイ．ウールゼイ
アメリカ合衆国 テキサス州プラノ，ミドルコーブ 608

(72)発明者 ジエラール ショベル
フランス国 アンティーブ，シュマン ドラ シュケット，レ ベルジェル ドバル コンスタンス